Implementation and Optimization Report of Adaptive Thresholding Methods Project

Zhiyuan Pan
Friedrich-Schiller-University Jena
zhiyuan.pan@uni-jena.de

Abstract. This project is an image processing application based on OpenCV and CMake, aiming to demonstrate how to use modern C++ technologies and libraries to perform image analysis tasks. The project design ideas mainly focus on three core functions: calculating the local mean and standard deviation of the image, calculating the threshold matrix based on the local mean and standard deviation, and using this threshold matrix to binarize the image. In addition, the project also demonstrates how OpenMP can be used to parallelize computations to improve performance.

Keywords: binarization, document image, thresholding method, OpenCV, Github.

1 Introduction

1.1 Theory Introduction:

The threshold value is determined by a combination of the mean value of the window's pixels, and a weighted sum of the global mean value of all pixels in the image, and the adaptive standard deviation. The formula is given by:

$$T = m_W - \frac{m_W^2 - \sigma_W}{(m_g + \sigma_W) \times (\sigma_{\text{Adaptive}} + \sigma_W)} \quad (1)$$

Where equation (1):

- T is the thresholding value.
- m_W is the mean value of the window's pixels.
- σ_W is the standard deviation of the window's pixels.
- m_g is the mean value of all pixels in the image.
- σ_{Adaptive} is the adaptive standard deviation of the window.

The adaptive standard deviation, $\sigma_{Adaptive}$ is calculated based on the difference between the window's standard deviation and the minimum standard deviation, scaled by the difference between the maximum and minimum standard deviations observed in the image:

$$\sigma_{Adaptive} = \frac{\sigma_W - \sigma_{min}}{\sigma_{max} - \sigma_{min}} \quad (2)$$

Where equation (2):

- σ_{min} is the minimum standard deviation value of all windows in the document.
- σ_{max} is the maximum standard deviation value of all windows in the document.

Based on this T values, the binarization process is defined in Equation (3).

$$\mathbf{I}(\mathbf{x},\mathbf{y}) = \begin{cases} \mathbf{b}\text{lack}, & \mathbf{i}(\mathbf{x},\mathbf{y}) < \mathbf{T_W}, \\ \mathbf{white}, & \mathbf{Else} \end{cases} \quad (3)$$

Where $\mathbf{I}(\mathbf{x},\mathbf{y})$) is the image and $\mathbf{i}(\mathbf{x},\mathbf{y})$ is the input pixel value of the image.

1.2 design Introduction:

Local Mean and Standard Deviation Calculation (**calcMeanAndStdDev**): This function calculates the mean and standard deviation of pixel intensities within a window for each pixel in the image.

- cv::Mat: A matrix data type used in OpenCV to store images.
- cv::integral: Computes the integral of an image. It's used here to quickly calculate the sum and sum of squares within a window.
- std::pow: A standard C++ function, not from OpenCV, used to calculate powers.
- cv::Mat::create: Allocates size and type for a cv::Mat object.
- std::max and std::min: Standard C++ functions to find the maximum or minimum of two values.

Threshold Matrix Calculation (**calcThresholdMatrix**): This function computes a threshold matrix based on local means and standard deviations as well as the global mean.

- cv::mean: Calculates the mean of all the pixel values in the image.
- cv::Mat::zeros: Creates a cv::Mat object and initializes all elements to zero.
- std::min_element and std::max_element: Standard C++ functions that return an iterator to the smallest and largest element in a range, respectively.
- cv::Mat::at: Accesses a specific element of the cv::Mat object. This is templated to handle different data types within the matrix.

Image Binarization (**binarizeImage**): This function creates a binary image based on the input image and a threshold matrix.

- cv::Mat::zeros: (as above) used here to initialize the binary image to all zeros.
- cv::Mat::at: (as above) used to access and set the binary value of each pixel.

Using OpenCV for such tasks is beneficial because:

- It provides optimized algorithms that can process images quickly and efficiently.
- It simplifies the implementation of complex algorithms with its high-level intuitive interfaces.
- It allows for parallel computing with built-in functions that can significantly speed up the

execution time when processing large images.

2 Project Build and Management

2.1 Build and Testing

The project employs CMake as its build system, facilitating cross-platform builds and simplifying the management of dependencies and compilation settings. Through the CMakeLists.txt file, the project can be effortlessly compiled and executed across different development environments, including Integrated Development Environments (IDEs) and command-line tools.

To ensure the quality and correctness of the code, the project utilizes the Catch2 framework for unit testing. The tests encompass validation of core functionalities, including the accuracy of local mean and standard deviation calculations, proper generation of threshold matrices, and the effectiveness of binarization processing. With Continuous Integration (CI) processes in place, these tests are automatically executed upon any update to the code repository, ensuring that code changes do not introduce errors.

2.2 Code Management and Continuous Improvement

The project's code is hosted on GitHub, with version control managed through Git. This approach not only facilitates code version management and team collaboration but also leverages GitHub's features for issue tracking, code review, and project progress management. Weekly updates to the code are uploaded to GitHub, guaranteeing that all participants have access to the latest developments and changes.

This structured approach to project management and testing ensures that the project adheres to high standards of quality and reliability. By integrating comprehensive testing and effective code management practices, the project is positioned for ongoing improvement and adaptation to new requirements and challenges.

3 Code Performance Optimization

During the development of this project, we implemented various strategies to optimize code performance, ensuring efficient completion of image processing tasks. Here are the primary optimization measures we adopted:

3.1 Avoiding Redundant Computations

After the initial run and successful image output, we noticed that the program's runtime was excessively long. To address this issue, we reviewed the code and discovered that certain calculations within the calcThresholdMatrix() function (such as sigmaMin, sigmaMax, and

sigmaRange) were placed inside the loop, leading to a significant amount of redundant computations. By moving these calculations outside the loop, we substantially reduced unnecessary computational overhead, thereby enhancing the program's execution efficiency.

3.2 Pre-allocating Memory

We utilized the cv::Mat::create method to pre-allocate memory for the mean and standard deviation matrices. The create method checks whether the matrix has already been allocated memory of the specified size and type. If not, it allocates new memory space; if suitable memory space already exists, it performs no operation. This approach prevents unnecessary memory allocation and deallocation, further boosting performance.

3.3 Parallelizing Loops with OpenMP

For parallel processing within function loops, we employed the OpenMP library. For instance, the directive #pragma omp parallel for collapse(2) schedule(dynamic, blockSize) allowed us to distribute iterations of the loop across multiple threads for parallel execution. The collapse(2) directive instructs the compiler to merge two nested loops into one large loop for parallel processing, while schedule(dynamic, blockSize) enables dynamic scheduling, allocating loop iterations to threads based on the block size (blockSize). This parallelization significantly accelerated processing speed, especially for large images or intensive computations.

3.4 Optimizing Compilation Instructions in CMakeLists

I also optimized the compilation instructions in the CMakeLists file. Depending on the compiler (such as GNU, Clang, and MSVC), I set different compilation options to maximize performance. For GNU and Clang, we used -O3 -march=native for advanced optimization and targeting the native architecture, while for MSVC, we applied /O2 for speed optimization. These compiler-level optimizations further reduced the program's runtime.

3.5 Results of Performance Enhancement

Through these comprehensive optimization measures, we achieved significant performance improvements. Initially, the runtime of the binarizeImage function was 0.317145 seconds. After parallel processing, reducing redundant calculations, and pre-allocating memory, the runtime was shortened to 0.112469 seconds. Finally, after optimizing compiler settings, the runtime was further reduced to 0.0348645 seconds. Thus, we not only enhanced the efficiency of image processing but also laid the groundwork for handling larger image datasets.

4 Final Results Presentation

In order to benchmark the project results, I compared them against the outcomes produced by OpenCV's adaptiveThreshold function using the same image.

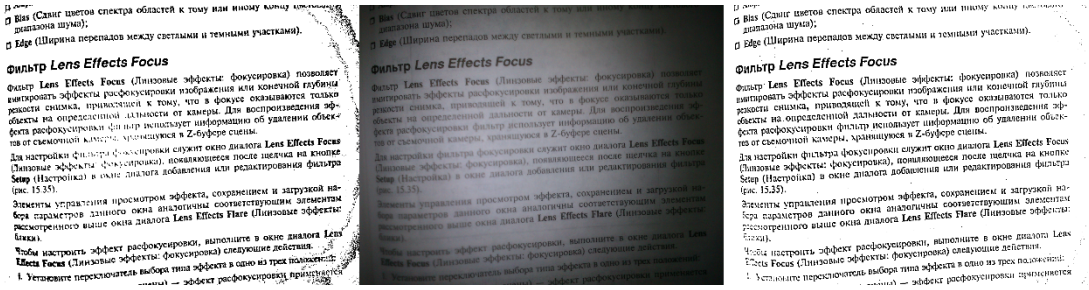

Figure 1. the results with my code & original image& with OpenCV

There were two significant differences in the results:
First, the central high-luminance part of the image was severely lost in my processing results, whereas the OpenCV's processed image was not noticeably affected by high luminance. Second, my results exhibited noticeable noise at the edges, likely due to issues in the algorithm I wrote for handling the window matrix, resulting in imperfect outcomes in low-luminance areas.

5 Conclusion

Through the implementation and optimization of Adaptive Thresholding Methods for Document Image Binarization in this project, I gained a profound experience and understanding of compiling and managing a complete C++ project using CMake. The use of a code repository also minimized the risk of data loss. The experience of optimizing loops and employing OpenMP for parallel processing has given me a clearer grasp of methods to enhance performance, correcting some of my tendencies for redundant definitions and calculations in code writing. Conducting comprehensive unit tests for a complete project has become part of my project development habit, benefiting all future coding workflows.

This project not only advanced my technical skills in image processing and software development but also underscored the importance of continuous learning and adaptation. The comparison with OpenCV's functionality provided valuable insights into potential areas for further improvement and innovation. Moving forward, the lessons learned from this project will inform more sophisticated and efficient solutions in the realm of image processing and beyond.

References

Bilal Bataineh, Siti N. H. S. Abdullah, K. Omar & M. Faidzul (2011). Adaptive Thresholding Methods for Documents Image Binarization. In M. Mendoza, & L.E. Sucar (Eds.), Pattern Recognition, MCPR 2011. Lecture Notes in Computer Science, vol 6718. (pp. 230-239). Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-21587-2_25